UN MODELO DE ENSEÑANZA- APRENDIZAJE MEDIANTE TÉCNICAS DE EVALUACIÓN COLABORATIVA EN CRÉDITOS PRÁCTICOS DE ASIGNATURAS DEL GRADO EN INGENIERÍA INFORMÁTICA

(ID10/147)

Memoria de Resultados

Convocatoria de Innovación Docente – Curso 2010-2011

Grupo de Innovación docente: Ana Belén Gil González (coordinadora), Ana de Luis Reboredo, Guillermo González Talaván, Sara Rodríguez González y Juan Francisco de Paz Santana

Departamento de Informática y Automática
Universidad de Salamanca - Facultad de Ciencias
Plaza de la Merced, s/n
37008 Salamanca

15 de Junio de 2011

CONTENIDO

FIGURAS

I. DATOS DEL PROYECTO

TÍTULO: Un Modelo de Enseñanza- Aprendizaje Mediante Técnicas de Evaluación Colaborativa en Créditos Prácticos de Asignaturas del Grado en Ingeniería Informática
REFERENCIA: ID10/147

PROFESOR COORDINADOR: Ana Belén Gil González

ORGANISMO: UNIVERSIDAD DE SALAMANCA
CENTRO: FACULTAD DE CIENCIAS

INVESTIGADORES QUE FORMAN EL EQUIPO:
Ana Belén Gil González
Ana de Luis Reboredo
Guillermo González Talaván
Sara Rodríguez González
Juan Francisco de Paz Santana

DURACIÓN: Octubre 2010 a junio 2011

II. INTRODUCCIÓN

La evaluación del aprendizaje de las competencias en el marco de la nueva ordenación de las enseñanzas universitarias oficiales ha de llevar parejo una revisión de los escenarios tradicionales de evaluación, motivada por un cambio en el paradigma del profesor como único responsable final de dicha tarea.

La evaluación por pares en el campo científico es una práctica usual que nos ha merecido la atención. Existe un desarrollo sobre el tema en campos específicos de la vida académica como es el de sistema de arbitraje de revistas especializadas y evaluación de proyectos de investigación.

Al mismo tiempo la necesidad de una evaluación continua de las competencias da lugar a un papel más activo de los alumnos en el proceso de aprendizaje, que permitirá integrar los procesos de evaluación como mecanismos que validen los logros de los estudiantes y al mismo tiempo que se canalicen como una guía motivadora en su proceso de aprendizaje.

En el grado de Ingeniero Informático, donde los créditos prácticos constituyen una parte fundamental de la carga docente; diseñar prácticas y realizar su posterior calificación es una tarea complicada, debido a que no existe una solución única y los criterios de evaluación de dichas prácticas constituyen en sí mismos una guía de entrenamiento de competencias. En este tipo de conocimiento las soluciones están relacionadas con su valided en el contexto, de manera que la evaluación de dichas prácticas no resulta fácil. Surge así la posibilidad de, en base a una serie prácticas relacionadas con la adquisición de competencias y su evaluación posterior, se pueda desarrollar un modelo de enseñanza-aprendizaje que permita al estudiante involucrarse de manera activa en la construcción del conocimiento de manera colaborativa.

En esta idea de redefinir los roles en el proceso de evaluación aparecen los métodos de aprendizaje colaborativo, que comparten la idea de que los estudiantes trabajen juntos para aprender y sean responsables a la vez del aprendizaje de sus compañeros tanto como del suyo propio, en el contexto del grado en Ingeniería Informática con la resolución tutorizada de prácticas evaluables mediante el criterio de pares.

Este planteamiento motiva la actuación del presente proyecto que ha servido para el diseño y validación de un modelo de evaluación de prácticas por pares que se incorporara a la construcción del conocimiento a través técnicas colaborativas de evaluación.

Para realizar dicho proyecto se conformó un equipo de trabajo que involucraba a un importante número de docentes relacionados con materias con una elevada carga de entregas de prácticas en el plan de estudios de Ingeniería Técnica Informática que se imparte en la Facultad de Ciencias de nuestra Universidad.

Una vez desarrollado este proyecto a lo largo del curso académico 2010/11, esta memoria hace un resumen final con las principales aportaciones y resultados obtenidos. Para ello en la

Sección 3 se presentarán los objetivos con los que parte el proyecto. La Sección 4 describirá el desarrollo y la generación de la metodología. A lo largo de la sección 5 se mostrarán los resultados obtenidos y se detallará el grado de consecución conseguido. Finalmente, la Sección 6 presentará las conclusiones de este proyecto de innovación docente.

Resulta también relevante acercarse al conjunto de anexos que incluye la memoria, donde se adjuntan materiales desarrollados para el desarrollo del proyecto piloto de puesta en marcha de la metodología en la asignatura de "Laboratorio de Sistemas Operativos" de ITIS.

III. OBJETIVOS

El objetivo principal del proyecto es el desarrollo de una metodología colaborativa que fortalezca los mecanismos tradicionales de aprendizaje en asignaturas con alto contenido práctico y desarrollo de actividades evaluables por los alumnos. Como objetivos generales se presentan:

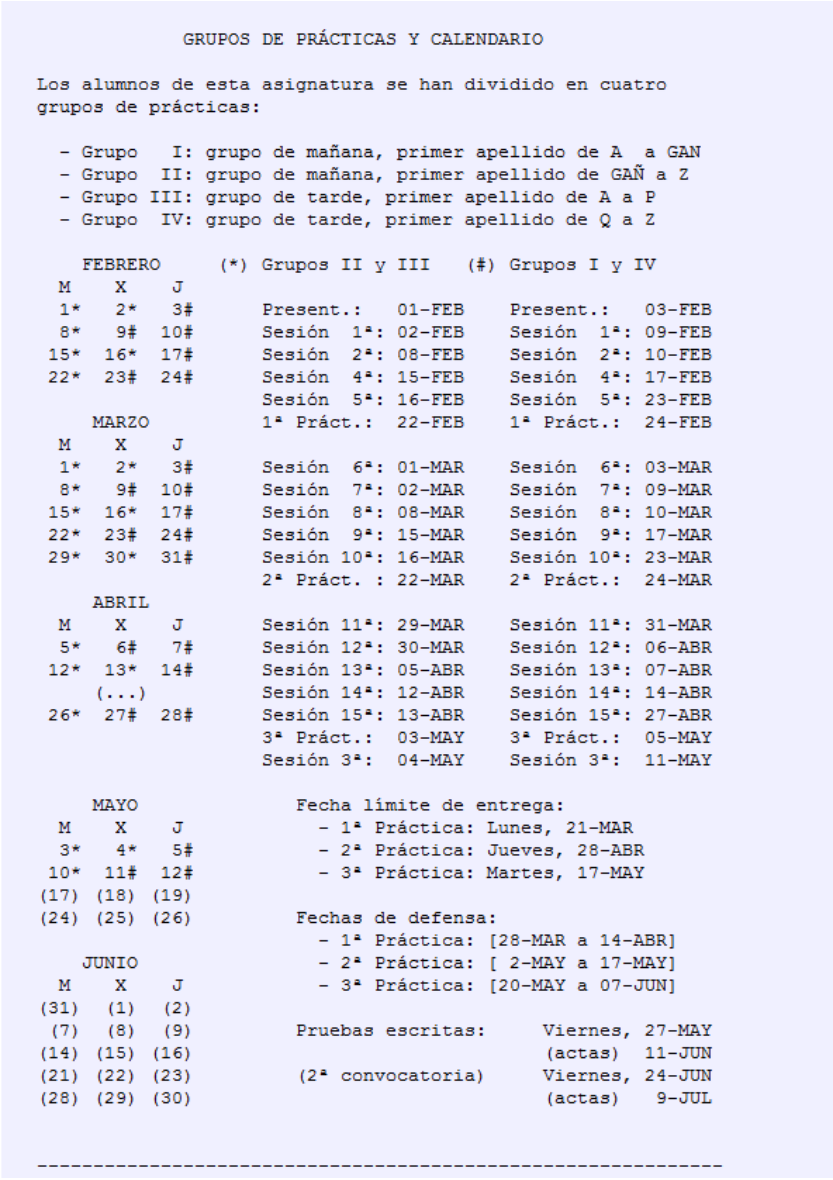
1. Estudio de técnicas de aprendizaje colaborativo en la evaluación continua en el aula
2. Desarrollo de técnicas de evaluación y coevaluación por pares en prácticas evaluables en titulaciones de ingeniería en Informática.
3. Involucrar a los alumnos en las tareas de evaluación y adquisición de competencias dentro de un modelo de resolución de prácticas.
4. Ponderar resultados diferenciados para distintas técnicas de evaluación de prácticas voluntarias planteados en distintas asignaturas del grado y la titulación.
5. Elaborar un modelo ejecución con los criterios de evaluación colaborativa de prácticas en asignaturas con alto contenido práctico para ingeniería informática que permita una metodología activa de enseñanza-aprendizaje en el aula.
6. Generación de materiales específicos para las prácticas de asignaturas del ámbito de “Sistemas Operativos” y similares para la implantación del grado en Ingeniero Informático.

IV. DESARROLLO DEL PROYECTO

Toda la información referente al funcionamiento y normas convenidas por los profesores en el proceso de evaluación de la asignatura son publicadas en la plataforma on-line de docencia empleada, bien en (Studium.usal.es) bien en la página web abierta de la asignatura.

A. METODOLOGÍA DE RESOLUCIÓN Y EVALUACIÓN DE LAS PRÁCTICAS

Desde el primer día de clase los alumnos disponen de un **calendario con todas las fechas** de (1) publicación de los enunciados de las prácticas, (2) entrega de prácticas y (3) defensas.



GRUPOS DE PRÁCTICAS Y CALENDARIO

Los alumnos de esta asignatura se han dividido en cuatro grupos de prácticas:

- Grupo I: grupo de mañana, primer apellido de A a GAN
- Grupo II: grupo de mañana, primer apellido de GAÑ a Z
- Grupo III: grupo de tarde, primer apellido de A a P
- Grupo IV: grupo de tarde, primer apellido de Q a Z

FEBRERO

M	X	J
1*	2*	3#
8*	9#	10#
15*	16*	17#
22*	23#	24#

MARZO

M	X	J
1*	2*	3#
8*	9#	10#
15*	16*	17#
22*	23#	24#
29*	30*	31#

ABRIL

M	X	J
5*	6#	7#
12*	13*	14#
(...)		
26*	27#	28#

MAYO

M	X	J
3*	4*	5#
10*	11#	12#
(17)	(18)	(19)
(24)	(25)	(26)

JUNIO

M	X	J
(31)	(1)	(2)
(7)	(8)	(9)
(14)	(15)	(16)
(21)	(22)	(23)
(28)	(29)	(30)

(*) Grupos II y III (#) Grupos I y IV

Present.:	01-FEB	Present.:	03-FEB
Sesión 1ª:	02-FEB	Sesión 1ª:	09-FEB
Sesión 2ª:	08-FEB	Sesión 2ª:	10-FEB
Sesión 4ª:	15-FEB	Sesión 4ª:	17-FEB
Sesión 5ª:	16-FEB	Sesión 5ª:	23-FEB
1ª Práct.:	22-FEB	1ª Práct.:	24-FEB
Sesión 6ª:	01-MAR	Sesión 6ª:	03-MAR
Sesión 7ª:	02-MAR	Sesión 7ª:	09-MAR
Sesión 8ª:	08-MAR	Sesión 8ª:	10-MAR
Sesión 9ª:	15-MAR	Sesión 9ª:	17-MAR
Sesión 10ª:	16-MAR	Sesión 10ª:	23-MAR
2ª Práct. :	22-MAR	2ª Práct.:	24-MAR
Sesión 11ª:	29-MAR	Sesión 11ª:	31-MAR
Sesión 12ª:	30-MAR	Sesión 12ª:	06-ABR
Sesión 13ª:	05-ABR	Sesión 13ª:	07-ABR
Sesión 14ª:	12-ABR	Sesión 14ª:	14-ABR
Sesión 15ª:	13-ABR	Sesión 15ª:	27-ABR
3ª Práct.:	03-MAY	3ª Práct.:	05-MAY
Sesión 3ª:	04-MAY	Sesión 3ª:	11-MAY

Fecha límite de entrega:
- 1ª Práctica: Lunes, 21-MAR
- 2ª Práctica: Jueves, 28-ABR
- 3ª Práctica: Martes, 17-MAY

Fechas de defensa:
- 1ª Práctica: [28-MAR a 14-ABR]
- 2ª Práctica: [2-MAY a 17-MAY]
- 3ª Práctica: [20-MAY a 07-JUN]

Pruebas escritas:	Viernes, 27-MAY	
	(actas)	11-JUN
(2ª convocatoria)	Viernes, 24-JUN	
	(actas)	9-JUL

Figura 1. Calendario de organización del Curso 2010/11 Para "Laboratorio de Sistemas Operativos"

El d**esarrollo de las prácticas se hará en parejas**. Para ello los alumnos forman los equipos agrupándose en parejas como quieran y a aquellos que no encuentran pareja el profesor les

asigna aleatoriamente compañero. Los alumnos indicarán al profesor los integrantes de equipo y dan un nombre al equipo formado. Cada alumno tendrá en el equipo un rol, capitán o grumete. El capitán será el interlocutor válido en el desarrollo de las prácticas para la comunicación con el profesor. Los equipos se mantienen a lo largo de todo el curso para la realización de las prácticas voluntarias. El profesor a partir del momento en el que están generados los equipos, realiza toda la publicidad de horarios, citaciones, etc. a través del nombre del equipo en unos casos a través de la plataforma studium y en otras a través de correo electrónico, etc.

```
DEFENSAS DE TERCERA PRÁCTICA
Martes 24 de Mayo, "Aula 4 de Informática":
9:30  yogiybubu:shabas:ortegaygaset
10:00 zamoranos:arminonly:hoyus
10:30 core2:duosacapuntas:charrococoches
11:00 r10d10:lakers:moupep
12:30 los_rulos:lasrubias
13:00 txusdani:parrillada:jdj
13:30 senecas:its_a_trap:maccabi


GENERACIÓN DE TRIADAS PARA LA SEGUNDA PRÁCTICA (29 de Abril de 2011)
arminonly:charrococoches:core2
duosacapuntas:hoyus:its_a_trap
jdj:lakers:lasrubias
losrulos:maccabi:moupep
ortegaygaset:parrillada:senecas
r10d10:shabas:txusdani
yogiybubu:zamoranos

DEFENSAS DE SEGUNDA PRÁCTICA (29 de Abril de 2011)

Martes 3 de Mayo, aula 1 de informática:
11:30 arminonly:charrococoches:core2
12:00 duosacapuntas:hoyus:its_a_trap
12:30 jdj:lakers:lasrubias
13:00 yogiybubu:zamoranos

Miércoles 4 de Mayo, aula SUN de informática (En el Dpto. de Informática):
12:00 ortegaygaset:parrillada:senecas
12:30 r10d10:shabas:txusdani
13:00 losrulos:maccabi:moupep
```

Figura 2. Histórico de organización de defensas de Prácticas a través de los nombre de equipos

B. INSTRUCCIONES DE LA PRÁCTICA Y ASPECTOS EVALUABLES

Una vez se propone una práctica, cada equipo genera una solución que se sube a la plataforma dispuesta en fecha y formato indicado (Ver Anexo I). Estas asociaciones cambiarán a lo largo de las 3 prácticas.

La defensa y evaluación de las prácticas de la asignatura consta de varias partes:

1. Cada grupo realizará la entrega de la resolución de la práctica planteada

2. Evaluación PREVIA a la defensa de la ejecución de las prácticas de los otros dos equipos.

Para la realización de esta revisión previa los equipos implicados se organizan para facilitar el trabajo colaborativo. Esta evaluación debe identificar en la resolución de las tres prácticas implicadas:

a. los aspectos críticos en la evaluación de la práctica
b. Si la ejecución funciona correctamente o no
c. de no funcionar adecuadamente alguna parte de la práctica, cómo se resolvería correctamente

Los alumnos disponen en Studium de un formulario de revisión para las prácticas. Cada formulario consta de unos apartados relativos a la práctica que cada grupo deberá rellenar y entregar firmado el día de la defensa. Se recomienda completar el formulario directamente en los espacios determinados para ello, imprimir después de rellenado y firmar

3. Se hace una defensa con todos los integrantes de los tres equipos implicados (6 persona en 3 equipos).
4. Durante la defensa cada equipo será cuestionado sobre aspectos relativos las prácticas de los otros 2 equipos y referenciada la información en el formulario generado para dicha práctica que entregarán relleno y firmado. Igualmente detallarán aspectos de su práctica que tendrán que defender razonadamente.
5. La nota de la práctica es sin embargo individual.

Otros criterios en la calificación:

- Si se detecta que la evaluación realizada de la práctica de los compañeros de asociación no es la correcta, se penalizará en la nota del equipo evaluador.
- Si por el contrario el equipo evaluador detecta y soluciona un error en la práctica de alguno de los equipos evaluados, se compensará positivamente en la nota del equipo evaluador.
- Aquel equipo cuya práctica tenía errores y estos fueron detectados por los equipos evaluadores es capaz de identificar y dar una posible solución a dichos errores durante la defensa, aligerarán la penalización de dicho error en la evaluación de su práctica.

La configuración de la triada cambia en cada práctica, de manera que no coincida ningún equipo a lo largo de los procesos de defensa. Esto permite fomentar el que un alumno pueda con 3 prácticas visualizar al menos la manera de trabajar de 6 equipos e interaccione con 12 personas de su grupo en el transcurso del curso.

A los alumnos se les indica el itinerario para realizar una práctica:

Pasos:
1. Formación de los grupos de prácticas
2. Entrega de la práctica en tiempo y forma

3. Generación de tríadas
4. Publicación de los nombres de los capitanes de los grupos
5. Puesta en contacto de los capitanes de las tríadas para el envío de las prácticas a cada grupo
6. Revisión por pares de la práctica: cada grupo revisará las prácticas de los otros dos grupos juntos con los que forma la tríada
7. Completar cada grupo el formulario de revisión para cada práctica evaluada (*)
8. Defensa
   8a. Entrega del formulario de revisión firmado
   8b. Defensa individual de la práctica

En el caso de estudio realizado en la asignatura de Laboratorio de Sistemas Operativos, la entrega de prácticas voluntarias supone un 30% de la nota final de la asignatura, con un 70% del examen escrito final. Las prácticas voluntarias propuestas fueron 3, que puntuaban hasta 1 punto sobre la nota final de la asignatura. En el caso de los grupos donde se siguió el protocolo de evaluación de defensa por pares las prácticas fueron valoradas de acuerdo a tres apartados:

Entrega de la práctica	HASTA 30%
Defensa personal	HASTA 50%
Revisión pares	HASTA 20%

V. RESULTADOS LOGRADOS

A. RESULTADOS

El proyecto ha permitido diseñar un modelo de evaluación colaborativa en la que el estudiante pueda ser evaluado de forma objetiva y sea partícipe en el proceso.

Esto ha permitido la redefinición del rol del alumno, que se hace responsable de la obtención del conocimiento en el desarrollo práctico de competencias propio y en el de sus compañeros.

Se ha trabajado en una metodología docente para asignaturas con un elevado número de alumnos y prácticas evaluables en los estudios de grado en Ingeniero Informático. Concretamente se ha realizado un **caso de estudio** en el aula para la asignatura de "Laboratorio de Sistemas Operativos" de 2º curso de ingeniero técnico en informática de Sistemas. Dicha asignatura, de carácter obligatorio, cuenta con 4.5 créditos, todos ellos prácticos, lo que la convertía en la idónea para realizar el estudio de la metodología. A lo largo del curso 2009/10 se realizó un primer acercamiento a la viabilidad de una metodología de evaluación por pares de las prácticas voluntarias y se vio que podría ser útil si se generaban herramientas que facilitasen la evaluación de los alumnos de las prácticas de sus compañeros. Las Figura 3 y Figura 4, muestran cómo en la primera convocatoria los grupos que realizaron una defensa de prácticas con metodología de triadas tenían un mayor número de aprobados (53.45%) en comparación con los grupos con defensa individual (38.96%). La segunda convocatoria no generaba apenas diferencias.

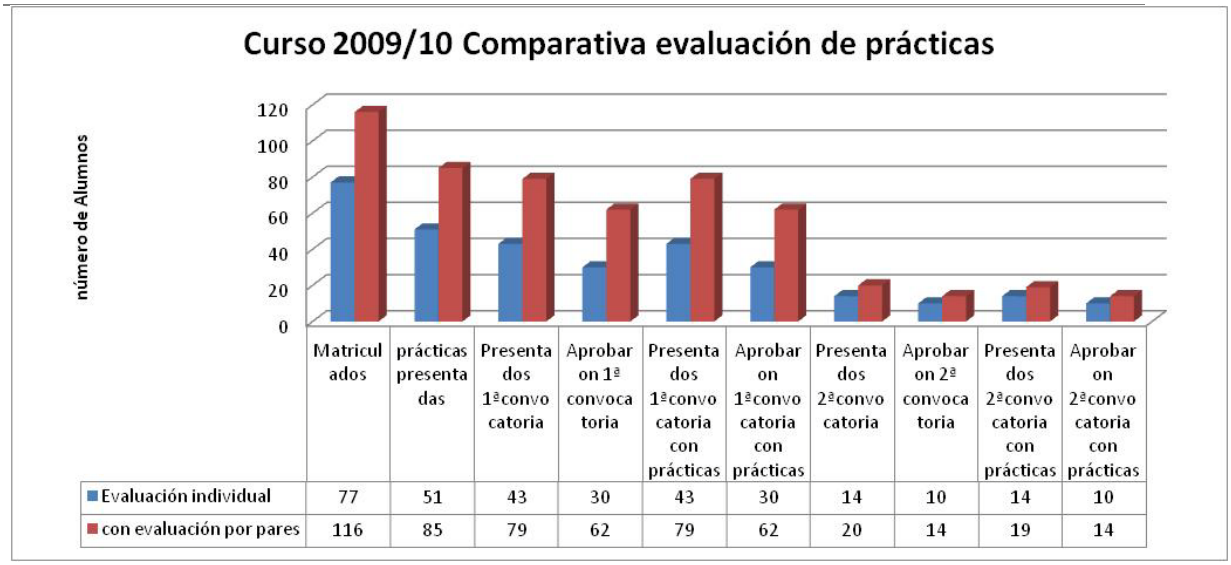


Figura 3. Gráfica comparativa de número de alumnos para tareas de evaluación realizadas (curso 2009/10)

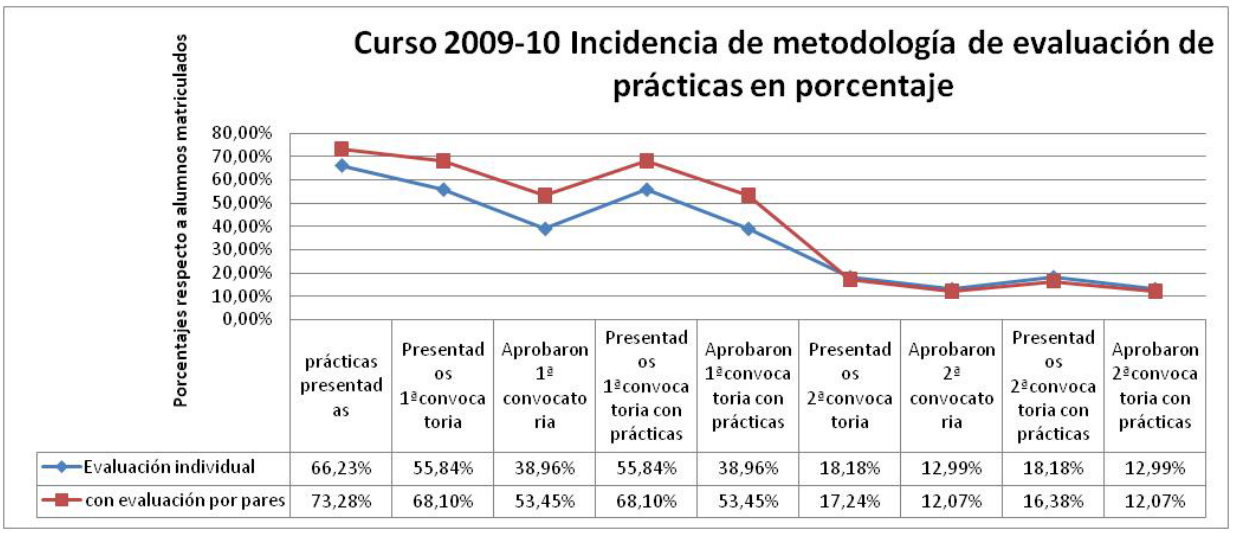
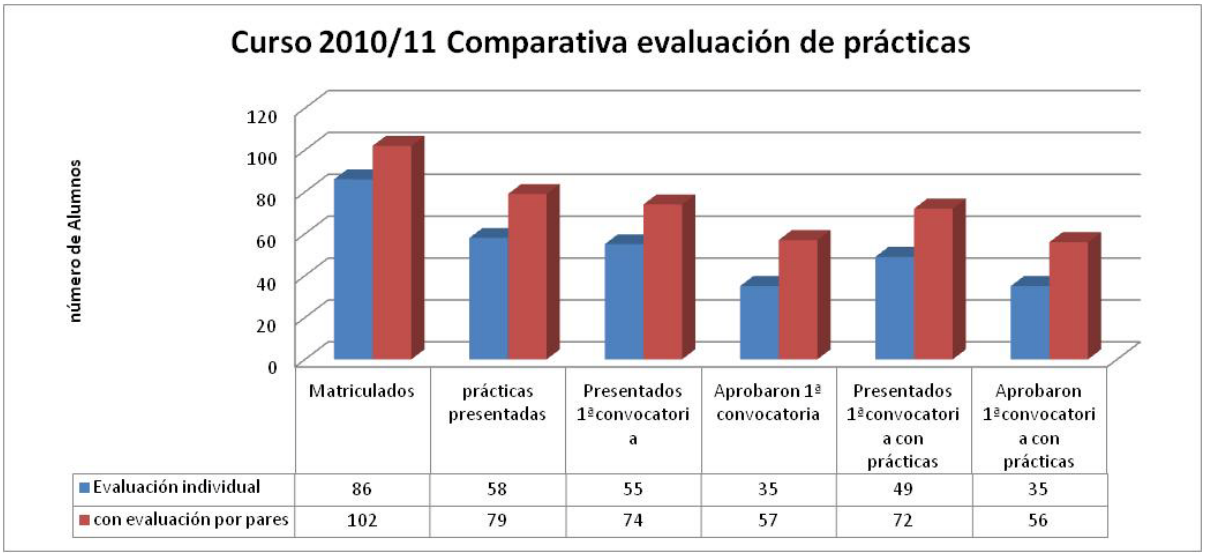


Figura 4. Gráfica comparativa en porcentaje respecto a alumnos matriculados para tareas de evaluación realizadas (curso 2009/10)

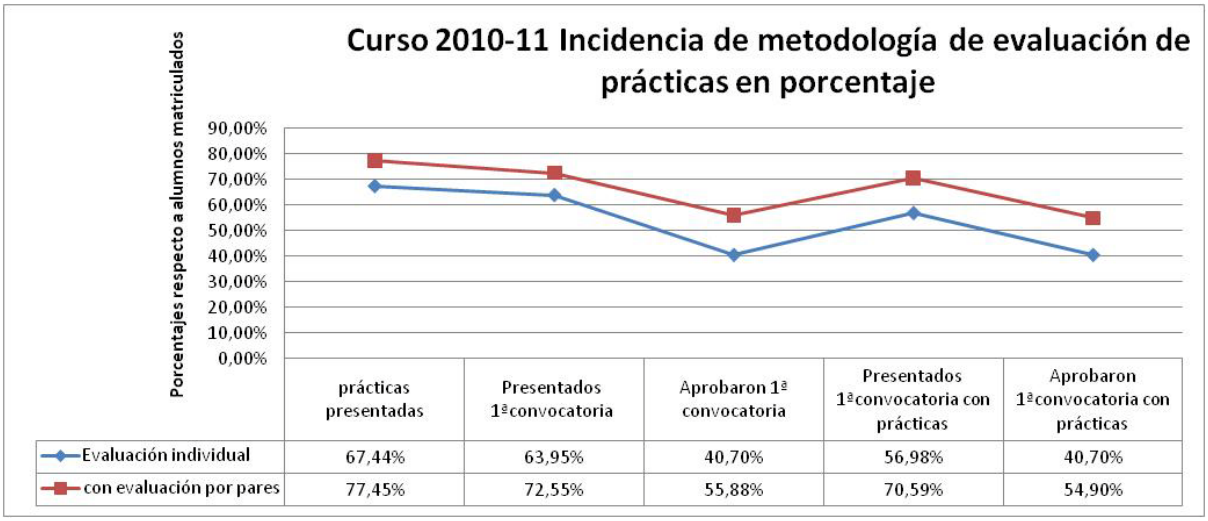
A lo largo del curso académico 20010/11 se generaron los grupos de trabajo para el desarrollo de una metodología docente. Se han elaborado materiales para poner en marcha un caso de estudio sobre la misma asignatura "Laboratorio de Sistemas Operativos" de 2º curso de ingeniero técnico en informática de Sistemas y seguir cotejar los valores obtenidos. Se ha estableciendo una metodología de evaluación de las prácticas por pares que se ha aplicado a los dos grupos de mañana mientras que para los dos grupos de la tarde se mantenía una defensa individual de las prácticas que permitiera su evaluación comparada. Aunque el desarrollo de las prácticas era el mismo para todos los grupos (mañana y tarde) a los alumnos de la mañana se les preparó para cada práctica un cuestionario, disponible en el Anexo 2. Dicho documento debía de ser rellenado por cada grupo en la evaluación de las dos prácticas de los equipos asignados y entregado el día de la cita fijada con los otros dos grupos en lo que denominamos "defensas por triadas".

Las Figura 5 y Figura 6 muestran los resultados obtenidos en la primera convocatoria de la asignatura. Nuevamente el porcentaje de alumnos que superan la asignatura es del (55.88%) en los grupos donde se puso en marcha la defensa por pares de las prácticas voluntarias, siendo de un 40.70% los grupos con defensa individual.



	Matriculados	prácticas presentadas	Presentados 1ªconvocatoria	Aprobaron 1ª convocatoria	Presentados 1ªconvocatoria con prácticas	Aprobaron 1ªconvocatoria con prácticas
Evaluación individual	86	58	55	35	49	35
con evaluación por pares	102	79	74	57	72	56

Figura 5. Gráfica comparativa de número de alumnos para tareas de evaluación realizadas (curso 2010/11)



	prácticas presentadas	Presentados 1ªconvocatoria	Aprobaron 1ª convocatoria	Presentados 1ªconvocatoria con prácticas	Aprobaron 1ªconvocatoria con prácticas
Evaluación individual	67,44%	63,95%	40,70%	56,98%	40,70%
con evaluación por pares	77,45%	72,55%	55,88%	70,59%	54,90%

Figura 6. Gráfica comparativa en porcentaje respecto a alumnos matriculados para tareas de evaluación realizadas (curso 2010/11)

Debido a que la presente convocatoria exige la entrega de memorias antes del 30 de Junio, no ha sido posible plasmar en esta memoria la incidencia de la metodología en la 2ª convocatoria de la asignatura.

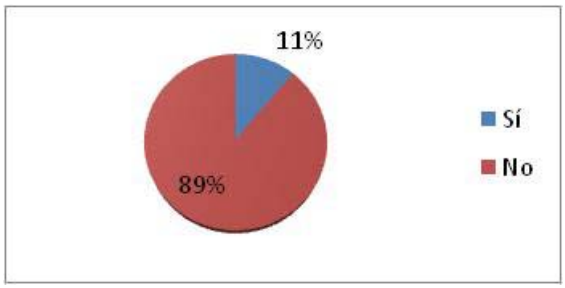
B. ENCUESTAS A LOS ALUMNOS

Para el conjunto de alumnos de la mañana se realizó al final de las clases, en el mes de mayo, una encuesta de evaluación. Dicha batería de cuestiones nos ha servido para conocer sus

impresiones sobre el mecanismo de revisión por pares y la defensa por triadas mediante el que han sido evaluadas sus prácticas. El cuestionario, colgado en la asignatura en Studium, fue contestado por 27 alumnos de un total de 86 matriculados en dicho grupo. Los resultados se muestran a continuación acompañados de un gráfico representativo para su mejor análisis.

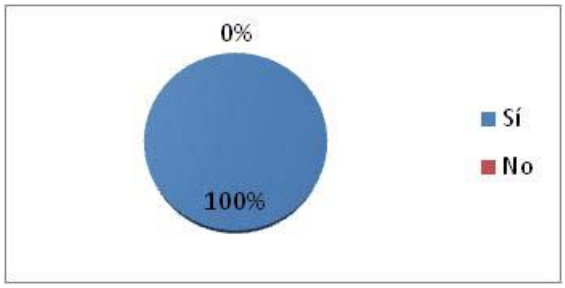
1. ¿Ha realizado en esta asignatura, Laboratorio de Sistemas Operativos, defensa de práctica en otros cursos académicos?

Sí	3
No	24
Total	27



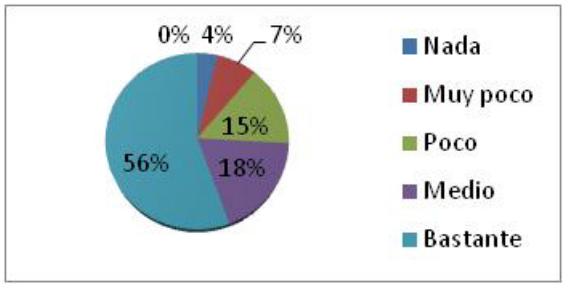
2. ¿Ha realizado en esta asignatura, Laboratorio de Sistemas Operativos, alguna defensa por triadas durante este curso académico 2010/2011?

Sí	27
No	0
Total	27



3. ¿Le ha resultado constructivo en el estudio de la asignatura el modelo de triadas seguido para la defensa de las prácticas?

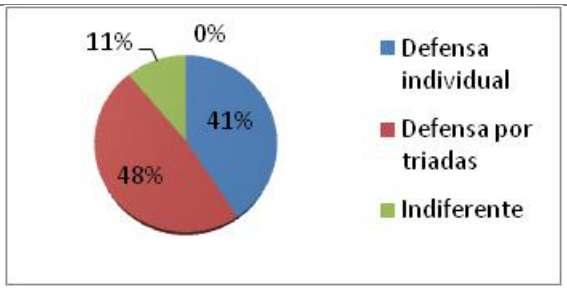
Nada	1
Muy poco	2
Poco	4
Medio	5
Bastante	15
Mucho	0



4. Si tuviese que elegir el

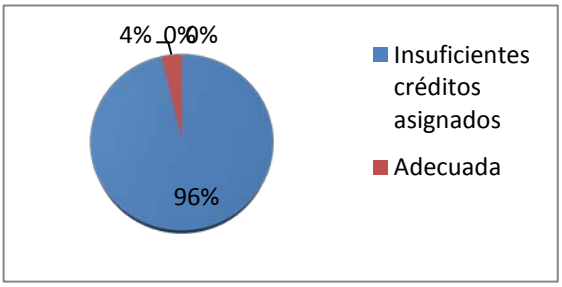
modelo de defensa para las prácticas de la asignatura, ¿Cuál elegiría?

Defensa individual	11
Defensa por triadas	13
Indiferente	3
NS/NC	0



5. Valore la relación entre la carga en créditos de la asignatura, 4.5, y la carga de trabajo/tiempo en la adquisición de los conocimientos presentados y la realización de actividades planteadas.

Insuficientes créditos asignados	26
Adecuada	1
Demasiados créditos asignados	0
NS/NC	0



6. Valore del 1 al 5, los distintos aspectos del desarrollo de la defensa de prácticas con triadas:

Satisfacción después de haber realizado este modelo de defensa	2,9
Desarrollo de las propias defensas en cada triada	2,8
Publicidad de las normas para las defensas	3

7. Aspectos negativos que destacaría del modelo de defensa de prácticas por triadas:

Alumnos valorando código no escrito por ellos
Demasiado trabajo delegado en el estudiante.
La asignatura ya tiene trabajo relación créditos
mucho tiempo en hacerlas.
ninguno
No se valoran correctamente
Poco tiempo para evaluar las practicas ajenas.
Preparar la defensa te lleva mucho tiempo .
Te enfrentas directamente a tus compañeros.

Tiempo para corregir otras prácticas
tiempo utilizado, el ambiente y la planificación
Preparar la defensa te lleva mucho tiempo .

8. Aspectos positivos que destacaría del método de defensa de prácticas por triadas:

Aprender de tus errores, por parte de compañeros
Aprender nuevas técnicas y detectar errores
aprendes distintas formas de hacer las practicas
Aprendes otras maneras de resolver los problemas.
Aprendizaje de otras formas de programar
aprendizaje conocimiento leer código de otro
Ayuda a mejorar una práctica realizada sin éxito.
comparar con otros compañeros la practica
Conocer la metodología usada por los otros grupos
Corregir tus errores a partir del código de otros
El ver errores y aciertos de otros
Entender distintos puntos de vista de solución
Hacer las practicas de diferentes formas
Las defensas son cortas.
Ninguno
Otras formas de hacer las cosas
poder subir nota con la evaluación a los demás
Posibilidad de aprender de los demás grupos.
POSIBILIDAD DE VER OTRAS PRACTICAS
Posible aprendizaje post defensa.
Puedes aumentar la puntuación
Puedes ver como lo han hecho otros.
se aprende de las cosas que tu no ves
Se aprende de los errores propios y de los demás
se aprende mucho mas
Se aprenden otros estilos de programación
Se ven otras soluciones al mismo problema
Si se hace bien, el aprender
Te permite obtener puntos para pasar la asignatura
Ver código de otra gente y no solo ver el tuyo
ver distintas maneras de programas
Ver el código de más gente a parte de tí ayuda
ves diferentes maneras de hacer lo mismo.
Ves distintas formas de hacer la practica
Ves el nivel y la forma de programación del resto.

C. GRADO DE CUMPLIMIENTO

Durante toda la duración del proyecto se ha realizado coordinadamente entre los profesores del equipo un estudio y elaboración de estrategias y metodologías docentes aplicadas para la enseñanza incluyendo la evaluación colaborativa por pares. Se han elaborado materiales de prácticas y diseños instruccionales de asignaturas individuales que han sido incluidas y gestionadas a través de las herramientas de docencia en línea, tanto Studium como en DiaWeb [1], estas últimas desarrolladas en el Departamento de informática y Automática. Para su evaluación en la práctica, se ha realizado una experiencia Piloto coordinando 4 grupos de prácticas, con 4 profesores y 188 alumnos en la asignatura "Laboratorio de Sistemas Operativos" de 2º curso de ingeniero técnico en informática de Sistemas. Dicha experiencia ha permitido generar a su vez nuevos materiales y ponderar el alcance de la metodología propuesta. En este sentido, los resultados son excelentes.

En suma y a pesar de las limitaciones temporales, se ha cubierto prácticamente todos los objetivos del proyecto. Una vez finalizado el proyecto los resultados incentivan a abordar objetivos que no estaban inicialmente previstos para la totalidad del proyecto y aplicar la metodología a más asignaturas en próximos cursos académicos.

Finalmente, el equipo de investigación está especialmente satisfecho con la labor de innovación puesta en marcha con la metodología de evaluación por pares en asignaturas con alto contenido en prácticas y número de alumnos. Debido a los buenos resultados obtenidos esta metodología permite asegurar una línea de investigación que merece la pena ser explorada en sucesivos proyectos con una sistematización de la metodología a diferentes asignaturas y ámbitos.

VI. CONCLUSIONES

El trabajo correspondiente al proyecto consistía en sentar las bases para poder desarrollar una metodología docente para asignaturas de ingeniería informática con alta carga práctica pero grupos numerosos. El elevado número de alumnos en este tipo de asignaturas hacen muy difícil el asumir la evaluación y el seguimiento requerido de los contenidos prácticos por parte del profesor. El EEES requiere un mayor número de profesorado que la situación económica actual no facilitará en los próximos años. El empleo de metodologías colaborativas en las que el alumno se integre en el sistema de evaluación de las actividades, siempre guiado y supervisado por el profesor ha generado en el caso de estudio unos buenos resultados. La metodología propuesta tiene como motor la resolución de prácticas voluntarias evaluables. Esto propicia que el alumno pueda reforzar los contenidos prácticos de las asignaturas con alta carga práctica con ayuda de sus compañeros y siempre en beneficio del grupo. El alumno

[1] http://diaweb.usal.es/

mediante la metodología planteada incrementa su actividad mediante el papel de solucionador de la práctica planteada y de evaluador de las de otros dos grupos de compañeros. Esto le permite una visión más global de los contenidos prácticos, pero también un incremento de tiempo tal y como han percibido claramente los alumnos y manifestado a través de las encuestas. A la vez y como contrapartida, asumen que aprenden de las soluciones de sus compañeros, corrigiendo las propias cuando no son óptimas o permitiendo a los demás ver soluciones más elaboradas. Tal y como se visualiza en la Figura 6 esta metodología ha generado un mayor número de presentados y aprobados de la asignatura con respecto a los grupos que siguieron una evaluación más individual.

Por otro lado, el proyecto ha permitido coordinar a distintos docentes que tienen asignada docencia en estas materias con alta carga práctica en estudios de informática toda la información de sus grupos. Esto ha permitido el despliegue de una metodología y su materialización en una asignatura concreta como caso de estudio.

La revisión del sistema de evaluación por pares y su aplicación a la metodología docente ha permitido la elaboración de materiales específicos para la materia Sistemas Operativos. Las entrevistas con los alumnos, entre los miembros del equipo y con compañeros de otras universidades, han permitido que se disponga de distintos puntos de vista que permitirán sustentar y mejorar la metodología tanto en esta materia con en otras.

Las experiencias obtenidas en cuanto a la innovación en la elaboración de una metodología de aprendizaje a través de evaluación por pares han sido excelentes y su incidencia en los resultados en el índice de alumnos que han aprobado la asignatura objeto del caso de estudio lo corrobora.

Todo lo anterior permite concluir que la línea abierta con este proyecto de innovación docente tiene excelentes perspectivas de futuro para su incorporación en asignaturas de grado en ingeniero informático para los próximos cursos. El proyecto ha generado directamente una acción de mejora en las asignaturas implicadas debido a que como resultado ha permitido la adecuación de:

- El diseño de las asignaturas contemplado en el Plan de Estudios,
- El programa de las asignaturas en la programación docente y
- El desarrollo y evaluación de la asignatura en la realidad docente.

Finalmente, se puede añadir que, a la vista de los resultados obtenidos y a todos los efectos, el proyecto ha sido desarrollado satisfactoriamente.

I. ANEXO I: ENUNCIADOS DE PRÁCTICAS

El presente anexo incluye el conjunto de prácticas voluntarias evaluables en la asignatura de Laboratorio de Sistemas Operativos durante el curso 2010/11, disponibles en la página web de la asignatura http://avellano.usal.es/~labssoo/.

A. PRIMERA PRÁCTICA EVALUABLE (2010-11): LOS ÁNGELES DE CHARLIE

1. ENUNCIADO.

El programa que hay que presentar constará de un único fichero fuente de nombre `charlie.c`. La correcta compilación de dicho programa, producirá un fichero ejecutable, cuyo nombre será obligatoriamente `charlie`. Respetad las mayúsculas/minúsculas de los nombres, si las hubiere.

La ejecución del programa producirá unos hechos basados en la conocida serie de televisión de finales de los años 70 del siglo pasado. En la ejecución participan seis personajes, representados por procesos: Charlie, Bosley, los tres ángeles (Sabrina, Jill y Kelly) y el malo.

Esto es lo que hace cada uno de los personajes:

1. **Charlie**: es el proceso padre. Crea el proceso de Bosley. Bosley le avisa cuando haya creado los procesos de los ángeles y, a continuación, crea el proceso del malo. Avisa a Bosley de que el malo ha sido creado.
2. **Bosley**: es el enlace entre Charlie y los ángeles. El proceso que lo representa es hijo del proceso de Charlie. Crea a los ángeles, avisa a Charlie de que han sido creadas y espera a que Charlie le avise de que ha creado al malo. Una vez el malo ha sido creado, avisa a los ángeles de que pueden comenzar a disparar sobre él.
3. **Los ángeles**: son tres. Al menos en esta práctica, no en la serie, son hijas de Bosley. Su vida consiste en disparar al malo, tratando de cargárselo. Comienzan a disparar cuando les avisa Bosley.
4. **El malo**: es malo, malísimo. Sigue la tradición de su familia: una larga saga que se remonta a tiempos prehistóricos. Alguno de ellos incluso llegó a participar en "Operación Triunfo", pero mejor correr un tupido velo. El malo es curioso en el sentido de que se reencarna. Su primera encarnación es como hijo de Charlie. Pasado un tiempo, el proceso tiene un hijo y se muere. Así lo hará veinte veces. Si alguno de los ángeles es capaz de matar cualquiera de esas reencarnaciones, habrán ganado. Si el malo llega a su reencarnación vigésima, los ángeles habrán perdido.

El modo en que los ángeles pueden disparar al malo es el que sigue. Enviarán una señal SIGTERM a dicho sujeto. Debido a que el malo está continuamente reencarnándose el PID no es fijo. Para que los ángeles puedan conocer el PID del malo, se va a usar un fichero proyectado en memoria, cuyo nombre será `pids.bin`. El fichero tendrá cabida para 20 números enteros almacenados en formato binario

(cuatro bytes por cada entero). Al principio todos los enteros estarán a cero. Cuando se reencarna el malo, escribe su nuevo PID en una posición libre del fichero al azar. Cuando un ángel quiere disparar, elegirá una posición al azar del fichero. Si vale cero, significa que la pistola se ha encasquillado y no manda señal. Si vale distinto de cero, manda la señal al PID que ha leído. Si coincide que dicho PID es el de la reencarnación actual del malo, lo mata y acaba. Si no, continúa disparando.

Los ángeles disponen de tres disparos para tratar de acabar con el malo. Devolverán a Bosley un 0 si han acabado con el malo y un 1, si no lo han podido hacer. Bosley recogerá los códigos de retorno de los ángeles y, en función de ellos devolverá un valor diferente a Charlie. Charlie escribirá en la pantalla un mensaje final como los que siguen, en función del resultado de la misión:

- `CHARLIE: "El pAjaro volO. Ahora se pone tibio a daiquiris en el Caribe"`
- `CHARLIE: "Bien hecho, Sabrina, siempre fuiste mi favorita"`
- `CHARLIE: "Bravo por Jill"`
- `CHARLIE: "Kelly, donde pones el ojo, pones la bala"`
- `CHARLIE: "Kelly, mala suerte, tus compaNeras acertaron y tU, no"`
- `CHARLIE: "Jill, otra vez serA, te apuntarE a una academia de tiro"`
- `CHARLIE: "Sabrina, no te preocupes, seguro que la pistola estA mal"`
- `CHARLIE: "Pobre malo. Le habEis dejado como un colador... Sois unos Angeles letales"`

Los tiempos que transcurren entre las acciones de los procesos se simularán mediante `sleep`. Así, los ángeles emplean, al azar, entre 6 y 12 segundos antes de poder disparar. El malo vive entre 1 y 3 segundos, al azar, en cada reencarnación.

La invocación de la práctica se hará con un argumento opcional:

```
charlie [velocidad]
```

El argumento opcional podrá valer `normal` o `veloz`. Si no se especifica este argumento, se entiende que su valor es `normal`. La diferencia estriba en que, a velocidad veloz, no se debe ejecutar ninguna pausa por parte de los procesos, aunque se indiquen en el enunciado. Esto es, a esa velocidad, no se invoca a `sleep` nunca.

Todos los procesos informarán por la salida estándar acerca de qué están haciendo. El mensaje final de Charlie será uno de los indicados más arriba. Los otros posibles mensajes serán:

- `CHARLIE: "Bosley, hijo de mis entretelas, tu PID es` *x*`. Espero a que me avises..."`
- `CHARLIE: "Veo que los Angeles ya han nacido. Creo al malo..."`

- `CHARLIE: "El malo ha nacido y su PID es x. Aviso a Bosley"`
- `BOSLEY: "Hola, papA, dOnde estA mamA? Mi PID es x y voy a crear a los Angeles..."`
- `BOSLEY: "Los tres Angeles han acabado su misiOn. Informo del resultado a Charlie y muero"`
- `SABRINA: "Hola, he nacido y mi PID es x"` (lo mismo para las otras dos)
- `SABRINA: "Pardiez! La pistola se ha encasquillado"` (lo mismo para las otras)
- `SABRINA: "Voy a disparar al PID x"` (lo mismo para las otras)
- `SABRINA: "BINGO! He hecho diana! Un malo menos"` (lo mismo para las otras)
- `SABRINA: "He fallado. Vuelvo a intentarlo"` (lo mismo para las otras)
- `SABRINA: "He fallado ya tres veces y no me quedan mAs balas. Muero"` (lo mismo para las otras)
- `MALO: "JA, JA, JA, me acabo de reencarnar y mi nuevo PID es: x. QuE malo que soy..."`
- `MALO: "AY, me han dado... pulvis sumus, collige, virgo, rosas"`
- `MALO: "He sobrevivido a mi vigEsima reencarnaciOn. Hago mutis por el foro"`

Como veis, el malo, aunque muy malo, es un malo culto, que hasta sabe latín. No uséis otros mensajes diferentes a los anteriores y respetadlos íntegramente hasta la última coma, por si se usa, para la corrección, una herramienta automática.

Los procesos, al hacer un `ps` desde la línea de órdenes, deben mostrar su nombre, es decir: charlie, bosley, sabrina, kelly, jill y malo. Para lograr esto, haced que los procesos recién nacidos que lo necesiten hagan un `exec` al mismo ejecutable, pero variando `argv[0]` de modo acorde. Si necesitárais pasar información extra, usad más argumentos. Nada más iniciar la función `main` una comprobación del valor de `argv[0]` os puede servir para guiar al proceso a ejecutar el código que le corresponde.

Para que el buffer intermedio usado por `printf` no interfiera con la salida de los procesos, es importante usar `write` para la salida por pantalla en su lugar.

NÚMEROS ALEATORIOS

Esta no es una práctica de programación, sino que es necesario programarla y, por consiguiente, saber programar. Es por ello que a continuación se incluyen unas pistas para generar números aleatorios (al azar) en vuestros procesos y no se deja para que lo investiguéis por vuestra cuenta.

Los ordenadores son las máquinas más previsibles que os podéis encontrar. Para ellas, es muy difícil hacer algo al azar. Por eso, cuando quieren generar un número al azar, lo que hacen es usar una fórmula matemática que, partiendo de un número (la semilla) genera otro tratando

de que el nuevo se parezca poco al anterior. El número nuevo pasa a ser la nueva semilla para generar el siguiente.

Para obtener una ristra aleatoria se debe, pues, hacer dos pasos: primero, establecer la semilla inicial (**esto solamente se hace una vez**) y segundo, ir sacando números.

Para establecer la semilla inicial, la biblioteca de C usa la función `void srand(unsigned int semilla)`. Pero, ¿qué semilla ponemos? Si ponemos un número que se nos ocurra, los números al azar que obtendremos serán buenos, pero siempre los mismos cada vez que ejecutemos el programa. Para obtener números diferentes, se le suele pasar a `srand` algo que dependa del momento en que se lanza el programa. Algo muy típico es:

```
srand(time(NULL));
```

Cuando tenemos varios procesos, la cosa se complica. Primero, los procesos van a heredar la misma semilla del padre, por lo que todos los hijos sacarán los mismos números de su chistera. Si el proceso, hace un `exec` es aún peor, pues todo ocurre como si nunca se hubiera llamado a `srand`. Hay, por consiguiente, que hacer un `srand` en cada nuevo proceso una sola vez y después de haber hecho sus `exec`s, si es que los hace. Ahí no acaban nuestros problemas, pues si ponemos la fórmula de arriba, que tiene una resolución de segundos, dará la misma semilla a todos los procesos, pues se crean muy rápidos. Lo mejor es que la modifiquéis a vuestro gusto para que también incluya el PID del proceso que hace la llamada.

Una vez hemos iniciado el generador de números aleatorios, hay que obtener los propios números. La función que hace esto es `int rand(void)`. La dificultad estriba en que esta función nos devuelve un número al azar entre 0 y una macro llamada `RAND_MAX`, lo que es muy poco últil. Lo más normal es que queráis obtener un número entre a y b, ambos incluidos (por ejemplo, entre 1 y 6, para la tirada de un dado). Esta expresión logra vuestro objetivo:

```
a+(int)(rand()/(1.0+RAND_MAX)*(b-a+1))
```

FINALIZACIÓN ORDENADA

La práctica deberá acabar ordenadamente cuando el usuario pulse CTRL-C. Los procesos deben morir y el padre, una vez hayan muerto todos imprimirá la frase: "Programa interrumpido".

RESTRICCIONES

- Se deberán usar llamadas al sistema siempre que sea posible, a no ser que se especifique lo contrario.
- No está permitido usar la función de biblioteca `system`, salvo indicación explícita en el enunciado de la práctica.
- No se puede suponer que los PIDs de los procesos de una ristra van a aparecer consecutivos. Puestos en plan exquisito, ni siquiera podemos suponer que estarán ordenados de menor a mayor (puede ocurrir que se agoten los PIDs y se retome la cuenta partiendo de cero).

- No está permitido el uso de ficheros, tuberías u otro mecanismo externo para transmitir información entre los procesos, salvo que se indique en el enunciado.

PLAZO DE PRESENTACIÓN.

Hasta el lunes, 21 de marzo 2011, inclusive.

NORMAS DE PRESENTACIÓN.

Acá están.

LPES.

. Las tareas que tiene que realizar el programa son variadas. Os recomendamos que vayáis programándolas y comprobándolas una a una. No es muy productivo hacer todo el programa de seguido y corregir los errores al final. El esquema que os recomendamos seguir para facilitaros la labor se os muestra a continuación:

1. Haced un pequeño programa al que se le pase los argumentos que se especifican en el enunciado. Imprimid los argumentos para depurar y considerad las opciones de error al meterlos. Una vez controlados, comentad la depuración.
2. Crearemos primero el proceso del malo, pues tiene la vida más fácil de realizar. Hacemos el `fork`, cada proceso imprime quién es y los dejamos en `pause`. Al hacer un `ps` desde otro terminal, debemos observar a los dos procesos, uno hijo del otro.
3. Los dos procesos se llaman `charlie` en el punto anterior. Debemos lograr que el malo cambie su nombre a `malo`. Para ello, usaremos un truco. Haremos una llamada a una función de la familia `exec`, por ejemplo, `execl`, que ejecute el ejecutable `charlie`, pero con `argv[0]` igual a `malo`. Daos cuenta que, al hacer el `exec`, el proceso *olvida* todo y recomienza su ejecución. Lo tenemos que redirigir a una función para que haga las tareas que le son propias. Pero para eso, nos podemos valer del contenido de `argv[0]`. Moved el mensaje donde el malo dice que es el malo a dicha función y comprobad que todo va como debe ir.
4. Completad el funcionamiento del malo. Primero, duerme. Luego tiene un hijo y se muere. Así, veinte veces. Charlie debe esperar por la muerte del primer malo, para que no se quede zombie.
5. Hecho ya el malo. Ahora, que Charlie cree a Bosley. Este código tiene que venir antes de la creación de los malos, como dice en el enunciado. Lo hacemos ahora aunque venga antes. Bosley se queda esperando a que Charlie le avise de que ha creado el malo. Charlie le avisa cuando eso ocurre.
6. Toca el turno de crear a los ángeles. Bosley los crea y se ponen los mensajes y se avisa según se marca en el enunciado.
7. Charlie crea el fichero de PIDs y lo pone a cero. Para ver si se ha creado el fichero de un modo correcto, podéis usar la orden de la línea de órdenes od, que sirve para ver el contenido binario de un fichero. El fichero debe medir 80 bytes justos. Si no lo mide, muy probablemente lo habéis creado y escrito en modo de texto, no en binario.

8. El siguiente paso consiste en que el malo proyecte el fichero y vaya escribiendo su PID en un sitio vacío cada vez que se reencarna. Al final de la ejecución el fichero debe estar lleno con los PIDs que ha ido teniendo el malo.
9. Haced que los ángeles disparen a la señal de Bosley. Imprimid los mensajes adecuados según el resultado de los disparos.
10. Programad ahora que Bosley se quede esperando y que los ángeles le devuelvan el resultado de su misión. Una vez tenga todos, le devuelve un valor a Charlie, indicando el resultado global.
11. Acabad de pulid los detalles que faltan: que acabe bien cuando se pulsa CTRL-C, etc.
12. Probad el modo veloz. No os extrañe si en este modo siempre pasa lo mismo. Va a depender mucho del reparto de CPU. Lo que sí tiene que ocurrir es que los mensajes que aparecen tengan sentido.

I. **No se puede usar `sleep()` o similares para sincronizar los procesos. Hay que usar otros mecanismos.**

II. Sabéis que si usáis *espera ocupada* en lugares donde explícitamente no se haya dicho que se puede usar, la práctica está suspensa. No obstante, existe una variedad de espera ocupada que podríamos denominar *espera semiocupada*. Consiste en introducir una espera de algún segundo en cada iteración del bucle de espera ocupada. Con esto el proceso no consume tanta CPU como en la espera ocupada, pero persisten los demás problemas de la técnica del sondeo, en particular el relativo a la elección del periodo de espera. Aunque la práctica no estará suspensa si hacéis espera semiocupada, se penalizará en la nota bastante si la usáis. En conveniente que la evitéis.

III. Evitad, en lo posible, el uso de variables globales. Tenéis la posibilidad de declarar *variables estáticas*.

IV. Tened cuidado con el uso de `pause()`. Salvo en bucles infinitos de `pauses`, su uso puede estar mal. Mirad la solución a la práctica propuesta en la sesión quinta acerca de él o el siguiente LPE.

V. El programa que he hecho se para a veces. O en mi casa se para, pero en clase, no. O en clase sí, pero en casa, no.
Solución.

VI. ¿Qué hago cuando mi programa se desboca para no perjudicar el funcionamiento de la máquina?
Solución.

VII. Debéis tener cuidado con un efecto que se produce por el hecho de que los descriptores de fichero heredados comparten un único puntero de fichero. Si cualquiera de los procesos lo mueve con `lseek` el resto lo ve movido. Este efecto secundario hace que el siguiente código sea erróneo para tratar de bloquear el fichero:

VIII. `lseek(fd,0,SEEK_SET);`
IX. `lockf(fd,F_LOCK,0);`

La razón es que entre las dos instrucciones se puede perder la CPU y otro proceso nos puede mover el puntero que nosotros pensamos que está al principio. La solución pasa por no usar los descriptores heredados sino que cada hijo, al nacer haga algo similar a:

```
case 0: /* COdigo del hijo */
```

```
close(fd);   // Cerramos el descriptor heredado
fd=open(...  // Lo volvemos a abrir
```

X. En el LPE anterior es evidente que el open que hace el hijo no puede llevar O_TRUNC. Si lo lleva, cuandoquiera que nazca un hijo, borrará el fichero pudiendo pillar justo antes de una lectura, que no leería nada. Lo debe abrir solamente con el flag O_RDWR

XI. Los procesos malos, al nacer uno detrás de otro y ser adoptados por init, son de difícil control. Sería interesante poder matarlos al final o cuando se usa CTRL-C. Una solución consiste en enviar una señal al PID 0. Si se hace así, la señal la reciben todos los procesos de la misma familia. Es decir, que si la manda Charlie la recibirán él mismo y todos los demás, incluidos los malos. Gracias a BraveSparks por probarlo antes de poder incluir este LPE

B. SEGUNDA PRÁCTICA EVALUABLE (2010-11): UN DÍA EN LAS CARRERAS (DE COCHES)

1. ENUNCIADO.

En esta práctica vamos a simular, mediante procesos de UNIX, una carrera automovilística.

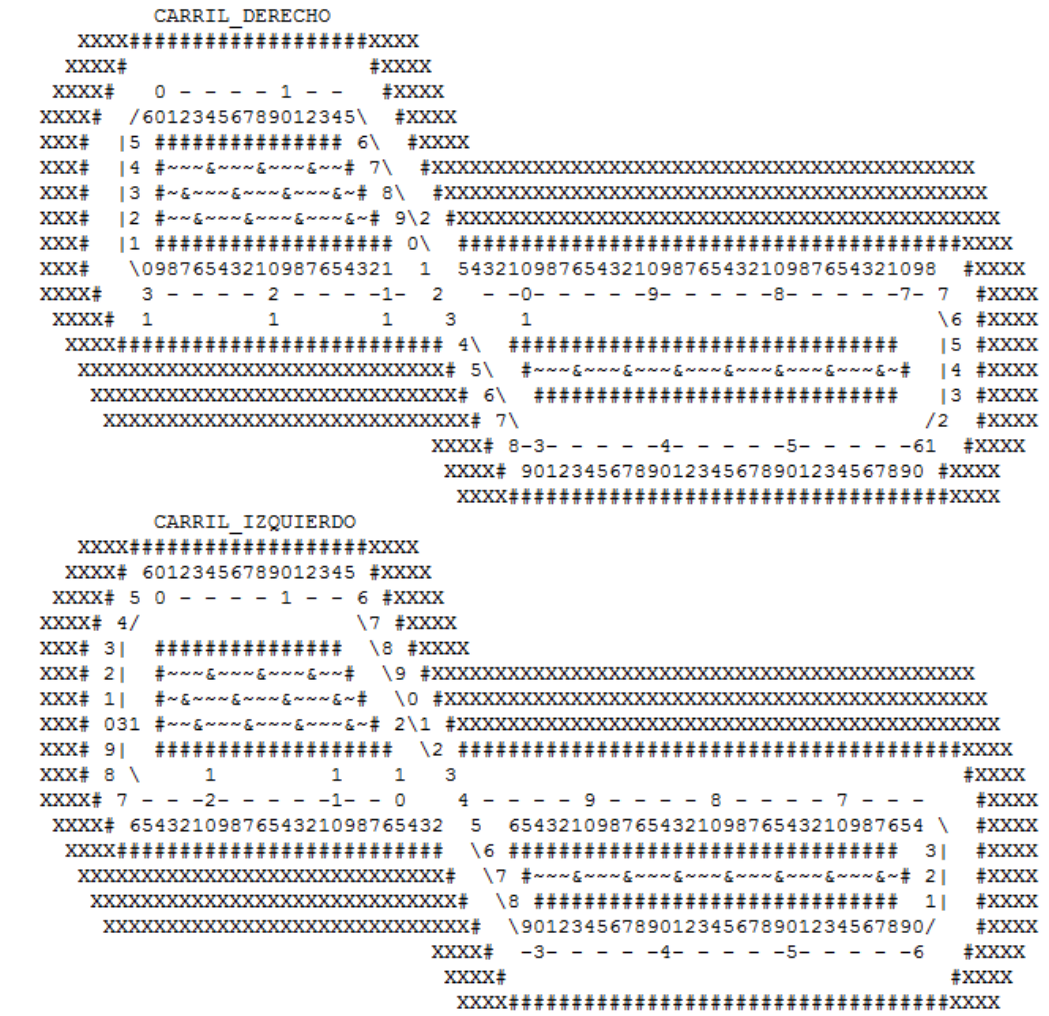
El programa constará de un único fichero fuente, `falonso.c`, cuya adecuada compilación producirá el ejecutable `falonso`. Respetad las mayúsculas/minúsculas de los nombres.

Para simplificar la realización de la práctica, se os proporciona una biblioteca estática de funciones (`libfalonso.a`) que debéis enlazar con vuestro módulo objeto para generar el ejecutable. Gracias a ella, muchas de las funciones no las tendréis que programar sino que bastará nada más con incluir la biblioteca cuando compiléis el programa. La línea de compilación del programa podría ser:

```
gcc falonso.c libfalonso.a -o falonso
```

Disponéis, además, de un fichero de cabeceras, `falonso.h`, donde se encuentran definidas las macros que usa la biblioteca.

El proceso inicial se encargará de preparar todas las variables y recursos IPC de la aplicación. También se encargará de crear todos los procesos que gobernarán los coches. Manejará así mismo, los semáforos del circuito. El circuito tiene forma de lemniscata y posee dos carriles por los que se circula en el mismo sentido. Los denominaremos carril derecho (CD) y carril izquierdo (CI). La posición de un coche en el circuito viene completamente determinada dando su carril y el desplazamiento dentro de él. El desplazamiento es un número entero comprendido entre 0 y 136, ambos incluidos. Sendos planos del circuito, uno para cada carril, con los desplazamientos indicados se pueden observar a continuación:



La práctica se invocará especificando dos parámetros exactamente desde la línea de órdenes. El primero es el número de coches que van a participar en la carrera. El segundo puede ser 0 ó 1. Si es 1, la práctica funcionará a velocidad normal. Si es 0, irá a la máxima velocidad.

Los coches, al principio, se pueden colocar como se desee, siempre que dos coches no compartan la misma posición. Una vez en movimiento, ningún coche podrá chocar con otro de la pista. La velocidad y los cambios de carril se dejan a vuestra discreción.

La función de cambio de carril hace que un coche cambie su carril y el desplazamiento según la siguiente tabla:

Der -> Izq		**Izq -> Der**	
0..13	0..13	0..15	0..15

14..28	15..29	16..28	15..27
29..60	29..60	29..58	29..58
61..62	60..61	59..60	60..61
63..65	61..63	61..62	63..64
66..67	63..64	63..64	67..68
68	64	65..125	70..130
69..129	64..124	126	130
130	127	127..128	130..131
131..134	129..132	129..133	131..135
135..136	134..135	134..136	136

Así, por ejemplo, un coche situado en (CD,80) pasa a (CI,75) al cambiar de carril. Uno que esté en (CI,126) pasa a (CD,130), por su parte.

Existe un cruce cerca del centro del circuito que está regulado mediante un par de semáforos. El funcionamiento de los semáforos lo realiza el proceso primero y deberá ser razonable (no se puede mantenerlos apagados o siempre en verde, o siempre en rojo, por ejemplo). Si un semáforo está en rojo, ningún coche deberá sobrepasarlo.

El circuito también cuenta con un contador automático situado en las posiciones (CD,133) y (CI,131). La aplicación desarrollada llevará cuenta, por su parte, en una variable compartida del número de pasos por el contador. Esto es, cada coche, al pasar por el contador, incrementará la variable compartida. Al finalizar el programa, se deberá pasar la dirección de memoria a la biblioteca para que esta compruebe que vuestra cuenta y la de ella coinciden. El programa estará en continua ejecución hasta que el usuario pulse las teclas CTRL+C desde el terminal. En ese momento, todos los coches deben parar (en un estado consistente) y morirse. El proceso primero esperará por su muerte. Informará a la biblioteca de que ha acabado, proporcionándole vuestra cuenta de vueltas y destruirá los mecanismos IPC utilizados.

Para que cualquier proceso pueda conocer en todo momento el estado del sistema, se va a usar una zona de memoria compartida. Los procesos no escribirán nunca en las partes controladas por la biblioteca. Son sólo informaciones. El mapa de la zona, expresado en bytes, es:

- 0-136: estado del carril derecho (' '<=>libre, otro valor <=> ocupado por un coche)
- 137-273: idem carril izquierdo
- 274: estado del semáforo para la dirección horizontal (ROJO, AMARILLO, VERDE ó NEGRO, macros de `falonso.h`)
- 275: idem para la dirección vertical
- 276-300: reservado biblioteca
- 301-: libre para vuestras necesidades, en particular para que contéis las vueltas

Siguiendo la misma filosofía, deberéis crear un array de semáforos, el primero de los cuales se reservará para el funcionamiento interno de la biblioteca. El resto, podéis usarlos libremente.

Para simplificar la realización de la práctica, se os proporciona una biblioteca estática de funciones (`libfalonso.a`) que debéis enlazar con vuestro módulo objeto para generar el ejecutable. Disponéis, además, de un fichero de cabeceras, `falonso.h`, donde se encuentran definidas las macros que usa la biblioteca. Las funciones proporcionadas por la biblioteca son las que a continuación aparecen. De no indicarse nada, las funciones devuelven -1 en caso de error:

- `int inicio_falonso(int ret, int semAforos, char *z)`
  El primer proceso, después de haber creado los mecanismos IPC que se necesiten y antes de haber tenido ningún hijo, debe llamar a esta función, indicando en `ret` si desea velocidad normal (1) o no (0) y pasando el identificador del conjunto de semáforos y el puntero a la zona de memoria compartida recién creados.
- `int inicio_coche(int *carril, int *desp, int color)`
  Esta función se debe invocar cuando se cree un nuevo coche. Se pasarán por referencia las variables para indicar el carril y desplazamiento del coche y por valor, su color. La variable `carril` podrá tener los valores `CARRIL_DERECHO` o `CARRIL_IZQUIERDO`, macros definidas en `falonso.h`. Los colores disponibles se encuentran definidos en el fichero de cabeceras `falonso.h`. No está permitido un coche azul porque no se ve. Podéis añadir 8 al código de color, si deseáis un tono más tenue o 16, si lo queréis más vivo. Si la posición donde queremos colocar el coche ya está ocupada, también se producirá un error.
- `int avance_coche(int *carril, int *desp, int color)`
  Dado un coche situado en la posición del circuito indicada por las variables enteras `carril` y `desp`, esta función le hará avanzar una posición en el circuito, modificando dichas variables de un modo acorde.
- `int cambio_carril(int *carril, int *desp, int color)`
  Igual que la función anterior, pero el coche se mantiene en la misma posición y sólo cambia de carril.
- `int luz_semAforo(int direcciOn, int color)`
  Pone el semáforo indicado en `direcciOn` al color señalado en `color` (ROJO, AMARILLO, VERDE o NEGRO).

- `int pausa(void)`
  Hace una pausa de aproximadamente una décima de segundo, sin consumir CPU.
- `int velocidad(int v, int carril, int desp)`
  Dado un coche situado en (`carril`, `desp`), y que marcha a una velocidad `v`, esta función ejecuta una pausa, sin consumo de CPU, del tamaño justo que hay entre dos avances del coche. La velocidad `v` estará comprendida entre 1 y 99, reservándose al valor 100 para la máxima velocidad que permite el ordenador (pausa efectiva nula).
- `int fin_falonso(int *cuenta)`
  Se llama a esta función después de muertos los hijos y haber esperado por ellos y antes de destruir los recursos IPC. El parámetro es la dirección de memoria compartida donde se ha llevado la cuenta de las vueltas de los coches (pasos por línea de meta).

Respecto a la sincronización interna de la biblioteca, se usa el semáforo reservado bien para poder actualizar la pantalla, bien para el manejo de la memoria compartida. Seguro que necesitaréis hacer sincronizaciones adicionales para que la práctica funcione. Para que esas sincronizaciones estén en sintonía con la biblioteca, os ofrezco ahora un seudocódigo de las funciones que realiza la biblioteca. S es semáforo interno que utiliza.

```
 * inicio_falonso:
     - limpia la pantalla
     - S=1
     - mensaje de bienvenida
     - dibuja el circuito

* inicio_coche:
     - comprobación de parámetros
     - W(S)
     - si posición ocupada, S(S), poner error, volver.
     - pintar el coche y actualizar la memoria compartida
     - S(S)

* avance_coche:
     - comprobación de parámetros
     - W(S)
     - borrar el coche y actualizar la memoria compartida
     - avanzar una posición
     - si hay choque, S(S), poner error, volver.
     - pintar el coche y actualizar la memoria compartida
     - si pasamos por el contador, incrementarlo y pintarlo
     - S(S)
     - si nos hemos saltado un semáforo en rojo, poner error, volver.

* cambio_carril:
     - comprobación de parámetros
     - W(S)
     - borrar el coche y actualizar la memoria compartida
     - cambiar de carril
     - si hay choque, S(S), poner error, volver.
     - pintar el coche y actualizar la memoria compartida
     - S(S)
     - si nos hemos saltado un semáforo en rojo, poner error, volver.

* luz_semAforo:
     - comprobación de parámetros
     - W(S)
     - dibujar semáforo y actualizar memoria compartida
     - S(S)

* fin_falonso:
```

```
        - si no coinciden las vueltas, poner error, volver.

    Cada vez que se pone un error, se pone W(S) y S(S) rodeando la impresión
en pantalla. Las funciones pausa() y velocidad() no usan el semáforo.
```

En cuanto al consumo de CPU, no debe consumirse CPU cuando un coche espere a que el semáforo de su carril se ponga en verde. Tampoco en los instantes iniciales cuando, después de haber colocado todos, deis el pistoletazo de salida. Podéis, aunque evidentemente se premiará al que logre hacerlo bien siendo como es difícil, consumir CPU para evitar choques o alcances o los coches que estén esperando a la cola de un semáforo en rojo. En estos casos, y para ser respetuosos con el ordenador, se realizará en todo caso "semiespera ocupada", intercalando en el sondeo una pausa de una décima de segundo.

En esta práctica no se podrán usar ficheros para nada, salvo que se indique expresamente. Las comunicaciones de PIDs o similares entre procesos se harán mediante *mecanismos IPC*.

Siempre que en el enunciado o LPEs se diga que se puede usar `sleep()`, se refiere a la *llamada al sistema*, no a la orden de la línea de órdenes.

Los mecanismos IPC (semáforos, memoria compartida y paso de mensajes) son recursos muy limitados. Es por ello, que vuestra práctica sólo podrá usar un conjunto de semáforos, un buzón de paso de mensajes y una zona de memoria compartida como máximo. Además, si se produce cualquier error o se finaliza normalmente, los recursos creados han de ser eliminados. Una manera fácil de lograrlo es registrar la señal SIGINT para que lo haga y mandársela uno mismo si se produce un error.

A) BIBLIOTECA DE FUNCIONES LIBFALONSO

Con esta práctica se trata de que aprendáis a sincronizar y comunicar procesos en UNIX. Su objetivo no es la programación. Es por ello que se os suministra una biblioteca estática de funciones ya programadas para tratar de que no tengáis que preocuparos por la presentación por pantalla, la gestión de estructuras de datos (colas, pilas, ...) , etc. También servirá para que se detecten de un modo automático errores que se produzcan en vuestro código. Para que vuestro programa funcione, necesitáis la propia biblioteca `libfalonso.a` y el fichero de cabecera `falonso.h`. La biblioteca funciona con los códigos de VT100/xterm, por lo que debéis adecuar vuestros simuladores a este terminal.

(1) FICHEROS NECESARIOS:

- `libfalonso.a`: para Solaris (ver 1.4), para el LINUX de clase (ver 1.4),
- `falonso.h`: Para todos.

2. PASOS RECOMENDADOS PARA LA REALIZACIÓN DE LA PRÁCTICA

Aunque ya deberíais ser capaces de abordar la práctica sin ayuda, aquí van unas guías generales:

1. Crear los semáforos, la memoria comparida y el buzón, y comprobad que se crean bien, con `ipcs`. Es preferible, para que no haya interferencias, que los definái privados.
2. Registrar SIGINT para que cuando se pulse ^C se eliminen los recursos IPC. Lograr que si el programa acaba normalmente o se produce cualquier error, también se eliminen los recursos (mandáos una señal SIGINT en esos casos).
3. Llamar a la función `inicio_falonso` en `main`. Debe aparecer la pantalla de bienvenida y, pasados dos segundos, dibujarse el circuito.
4. Probad a poner los semáforos a distintos colores.
5. Llega el momento de probar el circuito. Cread un hijo que maneje un coche que no cambie de carril y corra a velocidad constante.
6. Añadid otro coche y plantead una rutina que evite los choques por alcance.
7. Introducid los cambios de carril, si no lo habéis hecho ya para evitar los choques y cuidad de que no se produzcan choques al cambiar de carril.
8. Haced que el primer proceso maneje de modo razonable los semáforos.
9. Que los coches respeten el semáforo en rojo.
10. Tened en cuenta el argumento que indica el número de coches y cread tantos como indique dicho número.
11. Programar la cuenta, corregir la pulsación de CTRL+C si en alguna ocasión no funciona y completar la llamada a `fin_falonso()`.
12. Pulid los últimos detalles.

3. PLAZO DE PRESENTACIÓN.

Hasta el jueves 28 de abril de 2011, inclusive e improrrogable.

4. NORMAS DE PRESENTACIÓN.

Acá están. Además de estas normas, en esta práctica se debe entregar un esquema donde aparezcan los semáforos usados, sus valores iniciales y un seudocódigo sencillo para cada proceso con las operaciones *wait* y *signal* realizadas sobre ellos. Por ejemplo, si se tratara de sincronizar dos procesos C y V para que produjeran alternativamente consonantes y vocales, comenzando por una consonante, deberíais entregar algo parecido a esto:

```
SEMÁFOROS Y VALOR INICIAL: SC=1, SV=0.

SEUDOCÓDIGO:

        C                              V
       ===                            ===
  Por_siempre_jamás              Por _siempre_jamás
     {                              {
      W(SC)                          W(SV)
      escribir_consonante            escribir_vocal
      S(SV)                          S(SC)
      }                              }
```

5. EVALUACIÓN DE LA PRÁCTICA.

Dada la dificultad para la corrección de programación en paralelo, el criterio que se seguirá para la evaluación de la práctica será: si

a. la práctica cumple las especificaciones de este enunciado y,
b. la práctica no falla en ninguna de las ejecuciones a las que se somete y,
c. no se descubre en la práctica ningún fallo de construcción que pudiera hacerla fallar, por muy remota que sea esa posibilidad...

se aplicará el principio de "presunción de inocencia" y la práctica estará aprobada. La nota, a partir de ahí, dependerá de la simplicidad de las técnicas de sincronización usadas, la corrección en el tratamiento de errores, la cantidad y calidad del trabajo realizado, etc.

6. LPES.

I. ¿Dónde poner un semáforo? Dondequiera que uséis la frase, "el proceso debe esperar hasta que..." es un buen candidato a que aparezca una operación *wait* sobre un semáforo. Tenéis que plantearos a continuación qué proceso hará *signal* sobre ese presunto semáforo y cuál será su valor inicial.
II. Si ejecutáis la práctica en *segundo plano* (con ampersand (&)) es normal que al pulsar CTRL+C el programa no reaccione. El terminal sólo manda `SIGINT` a los procesos que estén en primer plano. Para probarlo, mandad el proceso a primer plano con `fg %` y pulsad entonces CTRL+C.
III. Un "truco" para que sea menos penoso el tratamiento de errores consiste en dar valor inicial a los identificadores de los recursos IPC igual a -1. Por ejemplo, `int semAforo=-1`. En la manejadora de `SIGINT`, sólo si `semAforo` vale distinto de -1, eliminái s el recurso con `semctl`. Esto es lógico: si vale -1 es porque no se ha creado todavía o porque al intentar crearlo la llamada al sistema devolvió error. En ambos casos, no hay que eliminar el recurso.
IV. Para evitar que todos los identificadores de recursos tengan que ser variables globales para que los vea la manejadora de `SIGINT`, podéis declarar una estructura que los contenga a todos y así sólo gastáis un identificador del espacio de nombres globales.
V. A muchos os da el error "Interrupted System Call". Mirad la sesión quinta, apartado quinto. Allí se explica lo que pasa con `wait`. A vosotros os pasa con `semop`, pero es lo mismo. De las dos soluciones que propone el apartado, debéis usar la segunda.
VI. A muchos, la práctica os funciona exasperantemente lenta en encina. Debéis considerar que la máquina cuando la probáis está cargada, por lo que debe ir más lento que en casa.
VII. A aquellos que os dé "Bus error (Core dumped)" al dar valor inicial al semáforo, considerad que hay que usar la versión de `semctl` de Solaris (con `union semun`), como se explica en la sesión de semáforos y no la de HPUX.
VIII. Al acabar la práctica, con CTRL+C, al ir a borrar los recursos IPC, puede ser que os ponga "`Invalid argument`", pero, sin embargo, se borren bien. La razón de esto es que habéis registrado la manejadora de `SIGINT` para todos los procesos. Al pulsar CTRL+C, la señal la reciben todos, el padre y los otros procesos. El primero que obtiene la CPU salta a su manejadora y borra los recursos. Cuando saltan los demás, intentan borrarlos, pero como ya están borrados, os da el error.
IX. El compilador de encina tiene un bug. El error típicamente os va a ocurrir cuando definái s una variable entera en memoria compartida. Os va a dar `Bus Error. Core`

dumped si no definís el puntero a esa variable apuntando a una dirección que sea múltiplo de cuatro. El puntero que os devuelve `shmat`, no obstante, siempre será una dirección múltiplo de cuatro, por lo que solo os tenéis que preocupar con que la dirección sea múltiplo de cuatro respecto al origen de la memoria compartida. La razón se escapa un poco al nivel de este curso y tiene que ver con el alineamiento de direcciones de memoria en las instrucciones de acceso de palabras en el procesador RISC de encina.

X. Se os recuerda que, si ponéis señales para sincronizar esta práctica, la nota bajará. Usad semáforos, que son mejores para este cometido.

XI. Todos vosotros, tarde o temprano, os encontráis con un error que no tiene explicación: un proceso que desaparece, un semáforo que parece no funcionar, etc. La actitud en este caso no es tratar de justificar la imposibilidad del error. Así no lo encontraréis. Tenéis que ser muy sistemáticos. Hay un árbol entero de posibilidades de error y no tenéis que descartar ninguna de antemano, sino ir podando ese árbol. Tenéis que encontrar a los procesos responsables y tratar de localizar la línea donde se produce el error. Si el error es "`Segmentation fault. Core dumped`", la línea os la dará si aplicáis lo que aparece en la sección Manejo del depurador. En cualquier otro caso, no os quedará más remedio que depurar mediante órdenes de impresión dentro del código.

Para ello, insertad líneas del tipo:

```
fprintf(stderr,"...",...);
```

donde sospechéis que hay problemas. En esas líneas identificad siempre al proceso que imprime el mensaje. Comprobad todas las hipótesis, hasta las más evidentes. Cuando ejecutéis la práctica, redirigid el canal de errores a un fichero con `2>salida`.

Si cada proceso pone un identificador de tipo "P1", "P2", etc. en sus mensajes, podéis quedaros con las líneas que contienen esos caracteres con:

```
grep "P1" salida > salida2
```

XII. Os puede dar un error que diga `Resource temporarily unavailable` en el `fork` del padre. Esto ocurre cuando no exorcizáis adecuadamente a los procesos hijos zombies del padre. Hay dos posibilidades para solucionarlo:

1. La más sencilla es hacer que el padre ignore la señal `SIGCLD` con un `sigaction` y `SIG_IGN`. El S.O. tradicionalmente interpreta esto como que no queréis que los hijos se queden zombies, por lo que no tenéis que hacer waits sobre ninguno de ellos para que acaben de morir
2. Interceptar SIGCLD con una manejadora en el padre y, dentro de ella, hacer los waits que sean necesarios para que los hijos mueran. Pero esto trae un segundo problema algo sutil: al recibirse la señal, todos los procesos bloqueados en cualquier llamada al sistema bloqueante (en particular, los WAITs de los semáforos) van a fallar. Si no habéis puesto comprobación de errores, los semáforos os fallarán sin motivo aparente. Si la habéis puesto, os pondrá `Interrupted system call` en el perror. Como podéis suponer, eso no es un error y debéis interceptarlo para que no ponga el perror y reintente el WAIT. La clave está en la variable `errno` que valdrá `EINTR` en esos casos.

XIII. No se debe dormir (es decir, ejecutar `sleeps` o `pausas`) dentro de una sección crítica. El efecto que se nota es que, aunque la práctica no falla, parece como si solamente un proceso se moviera o apareciera en la pantalla a la vez. Siendo más precisos, si dormís dentro de la sección crítica, y soltáis el semáforo para, acto seguido, volverlo a coger, dais muy pocas posibilidades al resto de procesos de que puedan acceder.

XIV. La biblioteca no reintenta los WAITs de su semáforo, por lo que, de recibirse una señal, podría fallar. Si os da problemas, simplemente ignorad la señal SIGCLD en el padre como se explica más arriba.

XV. El número de coches máximo no está fijado. Sin embargo, ninguna de las pruebas que se hagan para evaluar la práctica pasará de veinte coches.

XVI. Preguntáis acerca de dónde se hacen las comprobaciones de semáforos y el incremento de vueltas. Aquí lo tenéis:

- Para el semáforo vertical, los puntos de comprobación son (CD,21) y (CI,23). Es decir, los coches se deben parar en el (CD,20) y (CI,22).
- Para el semáforo horizontal, los puntos de comprobación son (CD,106) y (CI,99).
- Los incrementos de vueltas se efectúan cuando un coche avanza hasta (CD,133) ó (CI,131). Cuidado con los cambios de carril, no contéis vueltas de más.

Respecto al "pistoletazo" de salida, considerad que se trata de una sincronización de rendezvous o cita. Ningún coche se puede adelantar al pistoletazo y todos los coches tienen que estar iniciados cuando el pistoletazo se produzca. ¡A discurrir!

C. TERCERA PRÁCTICA EVALUABLE (2010-11): CORRE, CORRE, CORRE...

1. ENUNCIADO.

El programa propuesto constará de un único fichero fuente, `falonso2.cpp`, cuya adecuada compilación producirá el ejecutable `falonso2.exe`. Por favor, mantened los nombres tal cual, incluida la extensión. Se trata de realizar una práctica casi idéntica a la segunda práctica de este año, pero mediante un programa que realice llamadas a la API de WIN32.

Las principales diferencias respecto a la práctica segunda son:

- Se proporcionará una *Biblioteca de Enlazado Dinámico* (DLL) en lugar de la biblioteca estática `libfalonso.a`. La biblioteca se llamará `falonso2.dll`.
- Se realizará la práctica mediante hilos, uno para cada coche, además del hilo principal.
- En lugar del array de semáforos o buzones de mensajes se usarán los semáforos independientes u otros mecanismos de sincronización de WIN32 que se estimen convenientes.
- Aparece dos nuevas funciones de la biblioteca de prototipo:
  - `int FALONSO2_estado_semAforo(int direcciOn)`: Esta función devuelve el estado del semáforo que se le especifica como argumento (`HORIZONTAL` o `VERTICAL`). El estado devuelto será `ROJO`, `AMARILLO`, `VERDE` o `NEGRO`.

        - `int FALONSO2_posiciOn_ocupada(int carril, int desp)`: Verdadera, si la posición correspondiente al carril y desplazamiento está ocupada por un coche. Falsa, en caso contrario.
    - La práctica no funcionará indefinidamente hasta que se pulse CTRL+C, sino que durará 30 segundos, transcurridos los cuales, acabará, no sin antes comprobar que la cuenta de vueltas de la biblioteca coincide con la realizada por vuestro programa.

Debéis manejar la biblioteca de enlazado dinámico tal como se explica en la sesión decimoquinta. Disponéis asimismo del fichero de cabeceras `falonso2.h`. Las funciones proporcionadas por la DLL tienen prácticamente el mismo prototipo y funciónamiento que las de la práctica segunda, salvo:

- `int FALONSO2_inicio(int ret)`
  No es necesario pasar a esta función ningún array de semáforos o memoria compartida porque los hilos comparten ya de por sí la memoria.

El resto de funciones coinciden con las de la práctica citada, solo que anteponiento del prefijo FALONSO2_:

- `int FALONSO2_fin(int *cuenta);`
- `int FALONSO2_luz_semAforo(int direcciOn, int color);`
- `int FALONSO2_inicio_coche(int *carril, int *desp, int color);`
- `int FALONSO2_avance_coche(int *carril, int *desp, int color);`
- `int FALONSO2_velocidad(int v, int carril, int desp);`
- `int FALONSO2_cambio_carril(int *carril, int *desp, int color);`
- `int FALONSO2_pausa(void);`
- `void pon_error(const char *mensaje);`

Estad atentos pues pueden ir saliendo versiones nuevas de la biblioteca para corregir errores o dotarla de nuevas funciones. La sincronización interna de la biblioteca sigue los mismos esquemas expuestos en la práctica segunda.

A) CARACTERÍSTICAS ADICIONALES QUE PROGRAMAR

- El programa no debe consumir CPU apreciablemente en los modos de retardo mayor o igual que 1. Para comprobar el consumo de CPU, podéis arrancar el administrador de tareas de Windows mediante la pulsación de las teclas CTRL+ALT+SUPR. Observad, no obstante, que en las aulas de informática puede que esta opción esté deshabilitada.
- **IMPORTANTE**: Aunque no se indique explícitamente en el enunciado, parece obvio que se necesitarán objetos de sincronización en diferentes partes del programa.

B) BIBLIOTECA FALONSO2.DLL

Con estra práctica se trata de que aprendáis a sincronizar y comunicar entre sí hilos en WIN32. Su objetivo no es la programación. Es por ello que se os suministra una biblioteca dinámica de funciones ya programadas para tratar de que no tengáis que preocuparos por la presentación por pantalla, la gestión de estructuras de datos (colas, pilas, ...) , etc. También servirá para que se detecten de un modo automático errores que se produzcan en vuestro código. Para que vuestro programa funcione, necesitáis la biblioteca `falonso2.dll` y el fichero de cabeceras `falonso2.h`.

Esta biblioteca, aunque es para Windows, ha sido croscompilada desde Linux. Es por ello que puede dar algún problema al inicio y sea necesario actualizarla con nuevas versiones. Estad atentos, por si eso ocurre, a este apartado.

(1) FICHEROS NECESARIOS:

- `falonso2.dll` ver. 1.4: Descárgalo de aquí.
- `falonso2.h` ver. 1.4: Descárgalo de aquí.

(2) HISTORIA DE LAS VERSIONES

- 1.4: versión inicial

2. PASOS RECOMENDADOS PARA LA REALIZACIÓN DE LA PRÁCTICA

En esta tercera práctica, no os indicaremos los pasos que podéis seguir. El proceso de aprendizaje es duro, y ya llega el momento en que debáis andar vuestros propios pasos sin ayuda, aunque exista la posibilidad de caerse al principio.

3. PLAZO DE PRESENTACIÓN.

Hasta el martes, 17 de mayo de 2011, inclusive.

4. NORMAS DE PRESENTACIÓN.

Acá están. Además de estas normas, en esta práctica se debe entregar un esquema donde aparezcan los mecanismos de sincronización usados, sus valores iniciales y un seudocódigo sencillo para cada hilo con las operaciones realizadas sobre ellos. Por ejemplo, si se tratara de sincronizar con eventos dos hilos C y V para que produjeran alternativamente consonantes y vocales, comenzando por una consonante, deberíais entregar algo parecido a esto:

EVENTOS Y VALOR INICIAL: EC* (automático), EV (automático).

SEUDOCÓDIGO:

```
        C                                   V
       ===                                 ===
  Por_siempre_jamás                  Por _siempre_jamás
     {                                   {
      W(EC)                               W(EV)
      escribir_consonante                 escribir_vocal
      Set(EV)                             Set(EC)
      }                                   }
```

Debéis indicar, asimismo, en el caso de que las hayáis programado, las optimizaciones de código realizadas.

5. EVALUACIÓN DE LA PRÁCTICA.

Dada la dificultad para la corrección de programación en paralelo, el criterio que se seguirá para la evaluación de la práctica será: si

a. la práctica cumple las especificaciones de este enunciado y,
b. la práctica no falla en ninguna de las ejecuciones a las que se somete y,
c. no se descubre en la práctica ningún fallo de construcción que pudiera hacerla fallar, por muy remota que sea esa posibilidad...

se aplicará el principio de "presunción de inocencia" y la práctica estará aprobada. La nota, a partir de ahí, dependerá de la simplicidad de las técnicas de sincronización usadas, de las optimizaciones realizadas para producir mejores resultados, etc.

6. LPES.

I. No debéis usar la función `TerminateThread` para acabar con los hilos o `TerminateProcess` para acabar con los procesos. El problema de estas funciones es que están diseñadas para ser usadas solo en condiciones excepcionales y los hilos mueren abruptamente. Pueden dejar estructuras colgando, ir llenando la memoria virtual del proceso con basura o no invocar adecuadamente las funciones de descarga de la DLL.

II. Al ejecutar la práctica, no puedo ver lo que pasa, porque la ventana se cierra justo al acabar.

Para evitar esto, ejecutad la práctica desde el "Símbolo del sistema", que se encuentra en el menú de "Accesorios". **También es necesario que la ventana que uséis tenga un tamaño de 80x25 caracteres. Si no lo tenéis así, cambiadlo en el menú de propiedades de la ventana.**

III. Los caracteres de la práctica no aparecen correctamente.

La DLL puede usar algunos caracteres gráficos para la presentación. Para que funcionen correctamente, la ventana del "Símbolo del sistema" debe usar una

fuente que los incluya. Probad a cambiarla en la ventana de propiedades hasta hallar una correcta en vuestro sistema.

IV. Al ejecutar la función `LoadLibrary`, en lugar de aparecer la pantalla de presentación, aparece un mensaje que pone "`En DllMain`".

Es necesario que la ventana que uséis tenga un tamaño de 80x25 caracteres. Si no lo tenéis así, cambiadlo en el menú de propiedades de la ventana.

V. Cuando ejecuto la práctica depurando la pantalla se emborrona. ¿Cómo lo puedo arreglar?

Mejor depurad la práctica enviando la información de trazado escrita con `fprintf(stderr,...)` a un fichero, añadiendo al final de la línea de órdenes `2>salida`. De este modo, toda la información aparecerá en el fichero `salida` para su análisis posterior. No os olvidéis de incluir el identificador del hilo que escribe el mensaje.

VI. Tengo muchos problemas a la hora de llamar a la función XXXX de la biblioteca. No consigo de ningún modo acceder a ella.

El proceso detallado viene en la última sesión. De todos modos, soléis tener problemas en una conversión de tipos, aunque no os deis cuenta de ello. No se os dirá aquí qué es lo que tenéis que poner para que funcione, pues lo pondríais y no aprenderíais nada. Sin embargo y dada la cantidad de personas con problemas, a continuación viene una pequeña guía:

1. Primero debéis definir una variable puntero a función. El nombre de la variable es irrelevante, pero podemos llamarle igual que a la función (XXXX) por lo que veremos más abajo. Para definir el tipo de esta variable correctamente, debéis conocer cómo son los punteros a función. En la sesión quinta, se describe una función, `atexit`. Dicha función en sí ;no es importante para lo que nos traemos entre manos, pero sí el argumento que tiene. Ese argumento es un puntero a función. Fijándoos en ese argumento, no os resultará difícil generalizarlo para poner un puntero a funciones que admiten otro tipo de parámetros y devuelve otra cosa. Notad, además, que, al contrario que ocurre con las variables "normales", la definición de una variable puntero a función es especial por cuanto su definición no va solo antes del nombre de la variable, sino que lo rodea. Tenéis que poner algo similar a: `#$%&%$ XXXX $%&$@;`, es decir, algo por delante y algo por detrás.
2. Después de cargar la biblioteca como dice en la última sesión, debéis dar valor al puntero de función. Dicho valor lo va a proporcionar `GetProcAddress`. Pero, ¡cuidado!, `GetProcAddress` devuelve un FARPROC, que sólo funciona con punteros a funciones que devuelven `int` y no se les pasa nada `(void)`. Debéis hacer el correspondiente *casting*. Para ello, de la definición de vuestro puntero, quitáis el nombre, lo ponéis todo entre paréntesis y lo añadís delante de `GetProcAddress`, como siempre.
3. Ya podéis llamar a la función como si de una función normal se tratara. Ponéis el nombre del puntero y los argumentos entre paréntesis. Como os advertí más arriba, si habéis puesto XXXX como nombre al

puntero, ahora no se diferenciarán en nada vuestras llamadas a la función respecto a si dicha función no perteneciera a una DLL y la hubierais programado vosotros.

VII. Os puede dar errores en el fichero de cabecera `.h` si llamáis a vuestro fichero fuente con extensión `.c`. Llamadlo siempre con extensión `.cpp`.

VIII. Tened mucho cuidado si usáis funciones de memoria dinámica de `libc` (`malloc` y `free`). Son funciones que no *están sincronizadas*, es decir, no se comportan bien en entornos multihilo. O bien las metéis en una sección crítica o, mejor aún, tratad de evitarlas.

IX. En algunas versiones de Visual Studio os puede dar un error del tipo: `error XXXXX: 'FuncionW': no se puede convertir de 'const char[X]' a 'LPCWSTR'`. El motivo del error es que, por defecto, esa versión de Visual Studio supone que deseáis usar UNICODE (caracteres de 16 bits) en lugar de los normales (caracteres de 8 bits). La solución pasa por transformar el código fuente para que se ajuste a la programación en UNICODE de Microsoft o decirle a Visual Studio que no, que no queréis trabajar con UNICODE. Unos compañeros vuestros escriben diciendo que si en la configuración del proyecto seleccionáis "Juego de Caracteres->Sin establecer", se soluciona.

X. Pasos para que no dé problemas en Visual Studio 2008:

0. Es importante que seleccionéis, al crear un proyecto nuevo, "Aplicación de consola" de Win32
1. Dejad la función main tal cual la pone él: `int _tmain(int argc, _TCHAR* argv[])`
2. En las propiedades del Proyecto, propiedades de configuración, general, a la derecha, poned "Juego de caracteres", sin establecer.

Probad un "Hola, mundo" y, si no os da errores, los errores que aparezcan a continuación podéis confiar que son de que no estáis haciendo bien el acceso a la DLL. En general, o que no declaráis bien los punteros para acceder a las funciones o que no hacéis el correspondiente *casting* cuando llamáis a `GetProAddress`.

XI. Tenéis que incluir el fichero de cabeceras `windows.h` antes que el fichero de cabeceras de la DLL

II. ANEXO 2: DOCUMENTOS DE FORMULARIOS PARA LA EVALUACIÓN POR PARES

A continuación se incluyen los formularios que los grupos debían de rellenar una vez evaluados los equipos asignados para la evaluación por pares de cada una de las tres prácticas.

FORMULARIO PARA LA DEFENSA DE LA PRÁCTICA 1

LABORATORIO DE SISTEMAS OPERATIVOS Curso 2010/11

Grupo evaluador		
Nombre grupo	**Login capitán**	**Login grumete**
Datos capitán		
Nombre, Apellidos, DNI		**Firma**
Datos grumete		
Nombre, Apellidos, DNI		**Firma**

Grupo evaluado		
Nombre grupo	**Login capitán**	**Login grumete**

Criterios formulario

Éxito en ejecución (en encina)

Modo normal

Modo veloz

Salida por pantalla correcta

Modo normal

Modo veloz

Propuestas de mejora en secciones de la práctica:

Uso de señales, sincronización

Uso de variables (globales, locales, estáticas, etc.)
Uso de ficheros y proyecciones
Comprobación de errores
Finalización ordenada

Otros

FIRMAS GRUPO EVALUADOR:

FORMULARIO PARA LA DEFENSA DE LA PRÁCTICA 2

LABORATORIO DE SISTEMAS OPERATIVOS Curso 2010/11

Grupo evaluador		
Nombre grupo	**Login capitán**	**Login grumete**
Datos capitán		
Nombre, Apellidos, DNI		**Firma**
Datos grumete		
Nombre, Apellidos, DNI		**Firma**

Grupo evaluado		
Nombre grupo	**Login capitán**	**Login grumete**

Criterios formulario

Éxito en ejecución (en encina)

Posibles fallos de construcción que pudiera hacer fallar la ejecución

Propuestas de mejora en secciones de la práctica:
Sincronización (esquema de sincronización, simplicidad, buen uso de técnicas, etc.)

Uso de variables (globales, locales, estáticas, etc.)
Comprobación de errores
Finalización ordenada
Otros

FIRMAS GRUPO EVALUADOR:

FORMULARIO PARA LA DEFENSA DE LA PRÁCTICA 3

LABORATORIO DE SISTEMAS OPERATIVOS Curso 2010/11

Grupo evaluador		
Nombre grupo	**Login capitán**	**Login grumete**
Datos capitán		
Nombre, Apellidos, DNI		**Firma**
Datos grumete		
Nombre, Apellidos, DNI		**Firma**

Grupo evaluado		
Nombre grupo	**Login capitán**	**Login grumete**

Criterios formulario

Éxito en ejecución Consumo de recursos en tiempo de ejecución

Recursos Windows utilizados

Posibles fallos de construcción que pudiera hacer fallar la ejecución

Propuestas de mejora en secciones de la práctica:
Sincronización (esquema de sincronización, simplicidad, buen uso de técnicas, cierre de semáforos y recursos, etc.)

Uso de variables (globales, locales, estáticas, etc.)
Comprobación de errores

Finalización ordenada (espera por la muerte de los hilos o no, etc.)
Otros

FIRMAS GRUPO EVALUADOR: